```
  register int i,j;
  pas[0][0]=1;
  pas[1][0]=pas[1][1]=1;
  for(i=2;i<=n;i++)
  {
  pas[i][0]=1;
  for(j=1;j<i;j++)
      {
         pas[i][j]= pas[i-1][j-1]+pas[i-1][j];
      }
  pas[i][j]=1;
  }
}

main(void)
{
  pascals(10);
  int n,m;
  while(scanf("%d%d",&n,&m)!=EOF)
  {
  printf("%lu",pas[n][m]);
  }
  return 0;
}
```

Combination with repeated objects :

If in a word of s length character, a character is repeated l times, then the number of arrangement is:

$$\frac{s!}{l!}$$

If there is more then one repeated character the we can write,

$$\frac{s!}{l1!*l2!*.....ln!}$$

where,

l1, is the repeation times of first repeated character.
l2, is for second repeated character
and, so on…

but remember calculating both n! and l1!* l2! * l3!* ….. *ln may occurs overflow. So I use

$$\frac{s!}{l1!*l2!*\ldots\ln!} = \frac{1*2*3*\ldots s}{1*2*\ldots*l1*1*2*3*\ldots l2*\ldots*1*2*\ldots*\ln} = \frac{1}{1}*\frac{2}{2}*\ldots*\frac{m}{l1}*\frac{m+1}{1}*\frac{m+2}{2}*\ldots\frac{m2}{l2}*\ldots\frac{s}{\ln}$$

The code of this type of combination is as follows:

```
#include<stdio.h>
#include<math.h>
#include<string.h>

#define MAX 30

/* ************************************************************ */
/* A sample function that calculate how many ways that you can     */
/*  rearrange a word with its letter                          */
/* ************************************************************ */
double test(char *str)
{
     int de[MAX]={0};
     int ss[300] = {0};
     int l = strlen(str);
     int i,j=0;
     double c=1,d=1;
     for(i=0;i<l;i++)
     {
     ss[str[i]]++;
     if(ss[str[i]] > 1)
                  de[j++] = ss[str[i]];
     }
     c = 1;
     for(i=2;i<=l;i++)
     {
     c*=i;

     if(j>0)
                  d*= de[--j];
     if((d!=1) && !(fmod(c,d)))
     {
                  c /= d;
                  d=1;
     }
     }
     return c;
}

/* A sample main function */
```

```
main(void)
{
     char word[MAX];
     int n;
     int j=0;
     scanf("%d",&n);
     for(;n>0;n--)
     {
     scanf("%s",word);
     printf("Data set %d: %.0f",++j,test(word));
     putchar('\n');
     }
     return 0;
}
```

longest common subsequence (LCS)

Given two sequence X and Y, we say that a sequence z is a **common subsequence** of C and Y if Z is a subsequence of both X and Y.

longest common subsequence (LCS) is just the longest "**common subsequence**" of two sequences.

LCS_LENGTH(X,Y)

Input two sequence X<x1,x2,x3,....xm> and Y<y1,y2,y3.....yn>. It stores the C[i,j] values in the table C[0...m,0...n] whose entries are computed in row major order. It also maintain the table b[1...m,1...n] to simplify construction of an optimal solution.

```
1. m <- length[X]
2. n <- length[Y]
3. for i <- 1 to m
4.        do c[I,0] <- 0
5. for j <- 0 to n
6.        do c[0,j] <- 0
7. for i <- 1 to m
8.        for j <- 1 to n
9.        do if x_i = y_j
10.             then c[i,j] <- c[i -1,j-1]+1
11.                b[i,j] <- 1
12.             else if  c[i – 1,j] >= c[i,j-1]
13.               then c[i,j] <- c[i – 1,j]
14.                       b[i,j] <- 2
15.               else c[i,j] <- c[i,j-1]
16. return c and b
```

PRINT_LCS(b,X,i,j)

```
 1. if i = 0 or j = 0
 2.         then return
 3. if b[i,j] = 1
 4.         then PRINT_LCS(b,X,i-1,j-1)
 5. else if b[i,j] = 2
 6.         then PRINT_LCS(b,X,i-1,j)
 7. else PRINT_LCS(b,X,i,j-1)
```

Code for LCS

```
#include<stdio.h>
#include<string.h>

#define MAX 100 // size of each sequence

char str1[MAX],str2[MAX]; // the sequences

/* ********************************************** */
/*    This function print the LCS to the screen    */
/*     Input : A table generated by the LCS_LNT    */
/*   and the length ot the sequences       */
/*     Output: None                                 */
/* ********************************************** */

void p_lcs(int b[MAX][MAX],int i,int j)
{
  if ( (i == 0) || ( j == 0) )  return ;
  if( b[i][j] == 1 )
  {  p_lcs(b,i-1,j-1);
  printf("%3c",str1[i-1]);
  }
  else if( b[i][j] == 2)  p_lcs(b,i-1,j);
  else          p_lcs(b,i,j-1);
}

/* ********************************************* */
/*    This function calculate the LCS length */
/*   Input : Tow Sequence and an bool I. If     */
/*              I is FALSE(0) then the function   */
/*      do not print the LCS and if       */
/*   TRUE(1) then print using the  */
/*      above p_lcs function               */
/*   Output: None                  */
/* ********************************************* */
void LCS_LNT(bool I)
{
  int c[MAX][MAX]={0},b[MAX][MAX]={0},l1,l2;
  l1 = strlen(str1)+1;
  l2 = strlen(str2)+1;
```

```
  register int i,j;
  for(i=1;i<l1;i++)
  {   for(j=1;j<l2;j++)
      {   if( str1[i-1] == str2[j-1] )
          {   c[i][j] = c[i-1][j-1] + 1;
              b[i][j] = 1;
          }
          else if(c[i-1][j] >= c[i][j-1])
          {   c[i][j] = c[i-1][j];
              b[i][j] = 2;
          }
          else   c[i][j] = c[i][j-1];
      }
  }
  printf("%d\n",c[l1-1][l2-1]);
  if(I)  p_lcs(b,l1-1,l2-1);
}

/* a sample main function */
main(void)
{
  while(1)
  {
  if(!(gets(str1)))          return 0;
  if(!(gets(str2)))          return 0;
  LCS_LNT(1);
  }
}
```

Another code for finding LCS

```
#include<stdio.h>
#include<string.h>
#define MAX 105

char str1[MAX][50],str2[MAX][50],lcs[MAX][50];
int lcswl;

/* ********************************************** */
/*    This function print the LCS to the screen    */
/*     Input : A table generated by the LCS_LNT    */
/*          and the length ot the sequences           */
/*   Output: None                                  */
/* ********************************************** */
void p_lcs(int b[MAX][MAX],int i,int j)
{
  if ( (i == 0) || ( j == 0) )  return ;
  if( b[i][j] == 1 )
  {
  p_lcs(b,i-1,j-1);
  strcpy(lcs[lcswl++],str1[i-1]);
  }
  else if( b[i][j] == 2)           p_lcs(b,i-1,j);
```

```
  else                    p_lcs(b,i,j-1);
}

/* ***************************************** */
/*    This function calculate the LCS length     */
/*   Input : Tow Sequence and an bool I. If     */
/*            I is FALSE(0) then the function    */
/*      do not print the LCS and if              */
/*   TRUE(1) then print using the                */
/*      above p_lcs function                     */
/*   Output: None                                */
/* ***************************************** */
void LCS_LNT(int l1, int l2,bool I)
{
  int c[MAX][MAX]={0},b[MAX][MAX]={0};
  register int i,j;
  for(i=1;i<l1;i++)
  {   for(j=1;j<l2;j++)
      {   if( !(strcmp(str1[i-1],str2[j-1])))
          {   c[i][j] = c[i-1][j-1] + 1;
              b[i][j] = 1;
          }
          else if(c[i-1][j] >= c[i][j-1])
          {   c[i][j] = c[i-1][j];
              b[i][j] = 2;
          }
          else   c[i][j] = c[i][j-1];
      }
  }
  if(I)
  {
      lcswl = 0;
  p_lcs(b,l1-1,l2-1);
      j = c[l1-1][l2-1];
  printf("%s",lcs[0]);
      for(i = 1; i< j ;i++)         printf(" %s",lcs[i]);
  putchar('\n');
  }
}

/* Sample main function */
main(void)
{
  char word[50];
  int i=0,j=0,l1,l2,ln;
  while(scanf("%s",word)!=EOF)
  {
      ln = strlen(word);
  if(ln==1)
          if(word[0] == '#')
          {
              if(i==0)
              {
                  i = 1;
```

```
                l1 =j;
                j = 0;
                continue;
            }
            else
            {
                l2 = j;
                j = i = 0;
        test(l1+1,l2+1,1);
                continue;
            }
        }
  if(i==0)  strcpy(str1[j++],word);
     else   strcpy(str2[j++],word);
  }
  return 0;
}
```

Matrix Chain Multiplication (MCM)

Background

If two matrix A and B whose dimension is (m,n) and (n,p) respectively then the multiplication of A and B needs m*n*p scalar multiplication.

Suppose you are given a sequence of n matrix $A_1,A_2,.....A_n$. Matrix A_i has dimension (P_{i-1},P_i). Your task is to parenthesize the product $A_1,A_2,.....A_n$ such a way that minimize the number of scalar product.

As matrix multiplication is associative so all the parenthesizations yield the same product.

MATRIX_CHAIN_ORDER(P)

The input is a sequrnce P = <$P_0,P_1,P_2,.....P_n$> where length[P] = n+1. The procedure uses an auxlary table m[1....n,1....n] for storing the m[i,j] costs and auxlary table s[1...n,1...n] that records which index of k achive the optimal cost in computing m[i,j].

```
1.  n <- length[P]-1
2.  for i <- 1 to n
3.          do m[i,j] <- 0
4.  for l <- 2 to n
5.          do for i <- 1 to n - l + 1
6.               do j <- i + l - 1
```

```
 7.                        m[i,j] <- INF     //  INF is infnity
 8.                   for k <- i  to j -1
 9.            do q <- m[i,k]+m[k+1,j]+ P_{i-1}P_kP_j
 10.                  if q < m[i,j]
 11.                then m[i,j] <- q
 12.                        s[i,j] <- k
 13. return m,s
```

PRINT(s,i,j)

The following recursive procedure prints an optimal parentathesization of $(A_i,A_{i+1},.....Aj)$ given the s table computed by

MATRIX_CHAIN_ORDER and indices i and j. The initial call PRINT(s,1,n) prints optimal parentathesization of $(A_1,A_2,.....A_n)$.

```
1.   if i == j
2.       then print "A"i
3.       else  print "("
4.               PRINT(s,i,s[i,j])
5.      PRINT(s,s[i,j]+1,j)
6.               print ")"
```

Code :

```
#include<stdio.h>
#define MAX 15              /* number of matrix */
#define INF 4294967295          /* a maximum number of multiplication */

int num;
/* ********************************************** */
/*     Print out the sequence                     */
/*     Input : A two dimentional array(created    */
/*             by the matrix chan order function) */
/*             with there starting point          */
/*     Return : None                              */
/* ********************************************** */
void print(unsigned long s[][MAX],int i,int j)
{
  if(i==j)
  printf("A%d",num++);
  else
  {
  printf("(");
  print(s,i,s[i][j]);
  printf(" x ");
  print(s,s[i][j]+1,j);
  printf(")");
  }
}
```

```
/* ********************************************** */
/*   Find out the order                           */
/*   Input : A sequence of dimention and the      */
/*             number of matrix                      */
/*   Return : none                                 */
/* ********************************************** */

void matrix_chan_order(int *p,int n)
{
  unsigned long m[MAX][MAX] = {0};
  unsigned long s[MAX][MAX] = {0};
  unsigned int q;
  int l,j,i,k;
  for(l = 2; l <= n ;l++)
  {
      for(i = 1 ; i <= n - l +1 ; i++)
      {
          j = i + l -1;
          m[i][j] = INF;          for(k=i ; k < j ; k++)        {
              q = m[i][k] + m[k+1][j] + p[i-1]*p[k]*p[j];
              if(q < m[i][j])
              {
                  m[i][j] = q;
                  s[i][j] = k;
              }
          }
      }
  }
  num =1;
  print(s,1,n);
}

/* A sample main function */

main(void)
{

  int n;
  int p[MAX]={0};
  int i;

  while(scanf("%d",&n),n)
  {
  for(i=1;i<=n;i++)
         scanf("%d %d",&p[i-1],&p[i]);
  matrix_chan_order(p,n);
  putchar('\n');


  }
  return 0;
}
```

Big Number Addition

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#define MAX 1000
void reverse(char *from, char *to ){
     int len=strlen(from);
     int l;
     for(l=0;l<len;l++)
            to[l]=from[len-l-1];
     to[len]='\0';
}
void call_sum(char *first, char *sec, char *result){
     char F[MAX], S[MAX], Res[MAX];
     int f,s,sum,extra,now;
     f=strlen(first);
     s=strlen(sec);
     reverse(first,F);
     reverse(sec,S);
     for(now=0,extra=0;(now<f && now<s);now++){
            sum=(F[now]-'0') + (S[now]-'0') + extra;
            Res[now]=sum%10 +'0';
            extra= sum/10;
     }
     for(;now<f;now++){
            sum=F[now] + extra-'0';
            Res[now]=sum%10 +'0';
            extra=sum/10;
     }
     for(;now<s;now++){
            sum=F[now] + extra-'0';
            Res[now]=sum%10 +'0';
            extra=sum/10;
     }
     if(extra!=0) Res[now++]=extra+'0';
     Res[now]='\0';
     if(strlen(Res)==0) strcpy(Res,"0");
     reverse(Res,result);}

int main(){
     char fir[MAX],sec[MAX],res[MAX];
     while(scanf("%s%s",&fir,&sec)==2){
            call_sum(fir,sec,res);
            int len=strlen(res);
            for(int i=0;i<len;i++) printf("%c",res[i]);
            printf("\n");
     }
     return 0;
}
```

Big Number Subtraction

```
/*****          Big Number Subtraction          ******************/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#define MAX 1000
/*****************************************************************/
void reverse(char *from, char *to ){
     int len=strlen(from);
     int l;
     for(l=0;l<len;l++)
            to[l]=from[len-l-1];
     to[len]='\0';
}
int call_minus(char *large, char *small, char *result){
     char L[MAX], S[MAX];
     int l,s,now,hold,diff;
     l=strlen(large);
     s=strlen(small);
     bool sign = 0;
     if(l<s){
            strcpy(result,large);
            strcpy(large,small);
            strcpy(small,result);
            now=l; l=s; s=now;
            sign = 1;
     }//return 0;
     if(l==s){
            if(strcmp(large, small)<0){
                   strcpy(result,large);
                   strcpy(large,small);
                   strcpy(small,result);
                   now=l; l=s; s=now;
                   sign =1;
            }//return 0;
     }
     reverse(large,L);
     reverse(small,S);
            for(;s<l;s++)
            S[s]='0';
     S[s]='\0';
     for(now=0,hold=0;now<l;now++){
            diff=L[now]-(S[now]+hold);
            if(diff<0){
                   hold=1;
                   result[now]=10+diff+'0';
            }
            else{
                   result[now]=diff+'0';
                   hold=0;
            }
```

```
      }
      for(now=l-1;now>0;now--){
            if(result[now]!='0')
                  break;
      }
      result[now+1]='\0';
      reverse(result,L);
      strcpy(result,L);
      //return 1;
      return sign;
}
int main(){
      char fir[MAX],sec[MAX],res[MAX];
      while(scanf("%s%s",&fir,&sec)==2){
            if(call_minus(fir,sec,res)==1)
                  printf("-");
            int len = strlen(res);
            for(int i=0;i<len;i++)
                  printf("%c",res[i]);
            printf("\n");

      }
      return 0;
}
```

Big Number Multiplication

```
/******       Big Number Multiplication     *********************/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#define MAX 1000
/*****************************************************************/
void reverse(char *from, char *to ){
      int len=strlen(from);
      int l;
      for(l=0;l<len;l++)
            to[l]=from[len-l-1];
      to[len]='\0';
}
/*****************************************************************/
void call_mult(char *first,char *sec,char *result){
      char F[MAX],S[MAX],temp[MAX];
      int f_len,s_len,f,s,r,t_len,hold,res;
      f_len=strlen(first);
      s_len=strlen(sec);
      reverse(first,F);
      reverse(sec,S);
      t_len=f_len+s_len;
      r=-1;
```

```
     for(f=0;f<=t_len;f++)
            temp[f]='0';
     temp[f]='\0';
     for(s=0;s<s_len;s++){
            hold=0;
            for(f=0;f<f_len;f++){
                   res=(F[f]-'0')*(S[s]-'0') + hold+(temp[f+s]-'0');
                   temp[f+s]=res%10+'0';
                   hold=res/10;
                   if(f+s>r) r=f+s;
            }
            while(hold!=0){
                   res=hold+temp[f+s]-'0';
                   hold=res/10;
                   temp[f+s]=res%10+'0';
                   if(r<f+s) r=f+s;
f++;
            }
     }
     for(;r>0 && temp[r]=='0';r--);
     temp[r+1]='\0';
     reverse(temp,result);
}
/*************************************************************/
int main(){
     char fir[MAX],sec[MAX],res[MAX];
     while(scanf("%s%s",&fir,&sec)==2){
            call_mult(fir,sec,res);
            int len=strlen(res);
            for(int i=0;i<len;i++) printf("%c",res[i]);
            printf("\n");
     }
     return 0;
}
```

Big Number Division and Remainder

```
//*****         Big Number division         *********************//
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#define MAX 1000
/*********************************************************************/
int call_div(char *number,long div,char *result){
     int len=strlen(number);
     int now;
     long extra;
     char Res[MAX];
     for(now=0,extra=0;now<len;now++){
            extra=extra*10 + (number[now]-'0');
```

```
            Res[now]=extra / div +'0';
            extra%=div;
      }
      Res[now]='\0';
      for(now=0;Res[now]=='0';now++);
      strcpy(result, &Res[now]);
      if(strlen(result)==0)
            strcpy(result, "0");
      return extra;
}
/*****************************************************************/
int main(){
      char fir[MAX],res[MAX];
      long sec,remainder;
      while(scanf("%s%ld",&fir,&sec)==2){
            if(sec==0) printf("Divide by 0 error\n");
            else{
                  remainder=call_div(fir,sec,res);
                  int len=strlen(res);
                  for(int i=0;i<len;i++) printf("%c",res[i]);
                  printf("\t%ld",remainder);
                  printf("\n");
            }
      }
      return 0;
}
```

Big Number Square Root

```
//******       Big Number Sqrt        ************************//
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#define MAX 1000
/*****************************************************************/
int call_minus(char *large, char *small, char *result){
      char L[MAX], S[MAX];
      int l,s,now,hold,diff;
      l=strlen(large);
      s=strlen(small);
      if(l<s)
            return 0;
      if(l==s){
            if(strcmp(large, small)<0)
                  return 0;
      }
      reverse(large,L);
      reverse(small,S);
      for(;s<l;s++)
            S[s]='0';
```

```
     S[s]='\0';
     for(now=0,hold=0;now<l;now++){
            diff=L[now]-(S[now]+hold);
            if(diff<0){
                   hold=1;
                   result[now]=10+diff+'0';
            }
            else{
                   result[now]=diff+'0';
                   hold=0;
            }
     }
     for(now=l-1;now>0;now--){
            if(result[now]!='0')
                   break;
     }
     result[now+1]='\0';
     reverse(result,L);
strcpy(result,L);
     return 1;
}
/*************************************************************/
void call_sqrt(char *number,char *result,char *extra){
     int num,start,e,mul,l,r=0,len;
     char left[MAX],after[MAX];
     char who[5],temp[MAX],two[5];
     len=strlen(number);
     if(len%2==0){
            num=10*(number[0]-'0') + number[1]-'0';
            start=2;
     }
     else{
            num=number[0]-'0';
            start=1;
     }
     mul=(int) sqrt(num);
     result[0]=mul+'0';
     result[1]='\0';
     if(num-mul*mul ==0)
            extra[0]='\0';
     else
            sprintf(extra,"%d",num-mul*mul);
     for(;start<len;start+=2){
            e=strlen(extra);
            extra[e]=number[start];
            extra[e+1]=number[start+1];
            extra[e+2]='\0';
            two[0]='2';
            two[1]='\0';
            call_mult(result,two,left);
            l=strlen(left);
            for(mul=9;mul>=0;mul--){
                   who[0]=mul+'0';
                   who[1]='\0';
```

```
                    strcat(left,who);
                    call_mult(left,who,after);
                    if(call_minus(extra,after,temp)==1){
                           result[++r]=mul+'0';
                           result[r+1]='\0';
                           strcpy(extra,temp);
                           break;
                    }
else
                           left[l]='\0';
             }
      }
      result[++r]='\0';
}
/*****************************************************************/
int main(){
      char fir[MAX],ex[MAX],res[MAX];
      while(scanf("%s",&fir)==1){
             call_sqrt(fir,res,ex);
             int len=strlen(res);
             for(int i=0;i<len;i++) printf("%c",res[i]);
             printf("\n");
      }
      return 0;
}
```

Fibonacci Numbers

```
//* This run O(log n) time//
#include<stdio.h>
long conquer_fibonacci(long n){
      long i,h,j,k,t;
      i=h=1;
      j=k=0;
      while(n>0){
             if(n%2==1){
                    t=j*h;
                    j=i*h + j*k +t;
                    i=i*k + t;
             }
             t=h*h;
             h=2*k*h + t;
             k=k*k + t;
             n=(long) n/2;}
      return j;}
int main(){
      long n,res;
      while(scanf("%ld",&n)==1){
             res=conquer_fibonacci(n);
      printf("%ld\n",res);}
      return 0;
}
```

Fibnacci Number by Big Integer

```
#include<iostream.h>
#include<string.h>
int main()
{

        char *fibo[5001]={0};
        fibo[0]="0";
        fibo[1]="1";
        int l1=strlen(fibo[0]);
        int l2=strlen(fibo[1]);
        int l;
        for(long i=2;i<=5000;i++)
             {
              char str[10000];
              if(l1>=l2)l=l1;
              else   l=l2;
              int ca=0;
              long j,k,m,p;
              for(j=l1-1,k=l2-1,m=0,p=0;p<l;j--,k--,m++,p++)
               {
                   int s1;
                   if(j<0)fibo[i-2][j]='0';
                   s1=fibo[i-2][j]-48;
                   int s2;
                   if(k<0)fibo[i-1][k]='0';
                   s2=fibo[i-1][k]-48;
                   int ans=0;
                   ans+=s1+s2+ca;
                   if(ans>9)
                     {
                     str[m]=(ans-10)+48;
                     ca=1;
                     }
                    else
                     {
                     str[m]=ans+48;
                     ca=0;
                     }
                    }
              if(ca>0){str[m]=ca+48; m++;}
              str[m]='\0';
              fibo[i]=new char[m+1];
              long y=0;
               for(long x=m-1;x>=0;x--,y++)fibo[i][y]=str[x];
              fibo[i][y]='\0';
              l1=strlen(fibo[i-1]);
              l2=strlen(fibo[i]);

             }
```

```
        int n;

        while(cin>>n)
             {
             cout<<"The Fibonacci number for "<<n<<" is "<<fibo[n]<<"\n";
             }
        return 0;
}
```

BFS/DFS (Maze Problem)

Input :

```
8 8
#.#.#.#.
........
#.#.....
#.##.#..
..#..##.
#..##...
...#...#
#.s#d.#.
```

```
#include<stdio.h>
#include<conio.h>
#include<values.h>

#define N 100
#define MAX MAXINT

int mat[N][N], Q[N][3], cost[N][N], front = -1, rear = -1;
int m, n, sc, sr, p, nr, nc, r, c, leaf, i, j, res[10][10];
int R[4] = {0, -1, 0, 1};
int C[4] = {-1, 0, 1, 0};

void nQ(int r, int c, int p)
{
     Q[++rear][0] = r, Q[rear][1] = c, Q[rear][2] = p;}

void dQ(int *r, int *c, int *p)
{
     *r = Q[++front][0], *c = Q[front][1], *p = front;
}

void bfs(int sr, int sc, int p)
{
     nQ(sr, sc, p);
     do{
            dQ(&r, &c, &p);
            for(int i=0; i<4; i++)
            {
```

```
                    nr = r + R[i], nc = c + C[i];
                    if(mat[nr][nc]==1)
                    {
                          if(cost[nr][nc]>cost[r][c]+1)
                                cost[nr][nc]=cost[r][c]+1, nQ(nr, nc, p);
                    }
             }
      } while (rear!=front);
      leaf = p;
}
void show()
{
      for(int i=0; i<=m; i++)
      {
             for(int j=0; j<=n; j++)
             {
                    if(res[i][j])
                          printf("X");
                    else
                          printf(" ");
             }
             printf("\n");
      }
}

void dfs(int leaf)
{
      if(Q[leaf][2]==-1)
      {
             res[Q[leaf][0]][Q[leaf][1]] = 1;
             return;
      }
      dfs(Q[leaf][2]);
      res[Q[leaf][0]][Q[leaf][1]] = 1;
}

void main()
{
      clrscr();
      char ch;
      freopen("maze.txt", "r", stdin);
      scanf("%d%d", &m, &n);
      getchar();
      for(i=0; i<m; i++)
      {
             for(j=0; j<n; j++)
             {
                    cost[i][j] = MAX;
                    scanf("%c", &ch);
                    if(ch=='#')
                          mat[i][j] = 0;
                    else if(ch=='.')
                          mat[i][j] = 1;
                    else if(ch=='s')
```

```
                        mat[i][j] = 2, sc = j, sr = i;
                else
                        mat[i][j] = 3, res[i][j] = 1;
        }
        getchar();
    }
    bfs(sr, sc, -1);
    dfs(leaf);    show();
    }
```

MinSum (DFS/QUEUE)

Input :

```
5 6
3 4 1 2 8 6
6 1 8 2 7 4
5 9 3 9 9 5
8 4 1 3 2 6
3 7 2 1 2 3
```

Code

```
#include<stdio.h>
#include<conio.h>
#include<values.h>

#define N 100
#define MAX MAXINT

int mat[N][N], M[N][N], Q[N][4], front = -1, rear = -1;
int m, n, nc, nr, p, s, finalSum=MAX, leaf, r, c, i;

void init()
{
     for(int i=0; i<N; i++)
            for(int j=0; j<N; j++)
                   M[i][j] = MAX, mat[i][j] = 0;
}

void nQ(int r, int c, int p, int s)
{    Q[++rear][0] = r;   Q[rear][1] = c;     Q[rear][2] = p;
     Q[rear][3] = s;}
void dQ(int *r, int *c, int *p, int *s)
{
     *r = Q[++front][0];
     *c = Q[front][1];
     *p = front;
     *s = Q[front][3];
}
```